NORTH DAKOTA STATE UNIVERSITY

Aerial Camera for Nest Observation

Technical Report

Lucas Brendel
Travis Hettwer
Aaron Olson
Bre Schneider

5/3/2012

Table of Contents

Table of Tables

Table of Figures

Labeled Pictures of Device

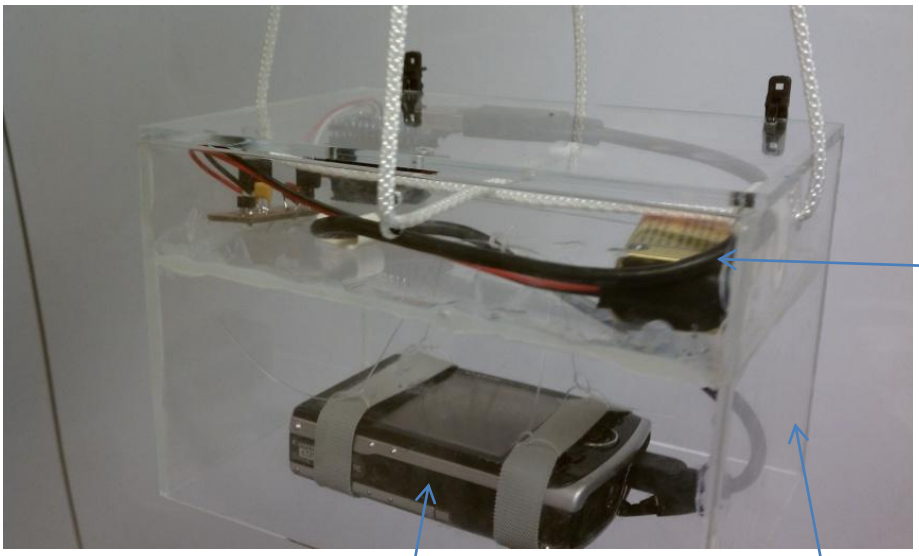


Figure 1: Enclosure

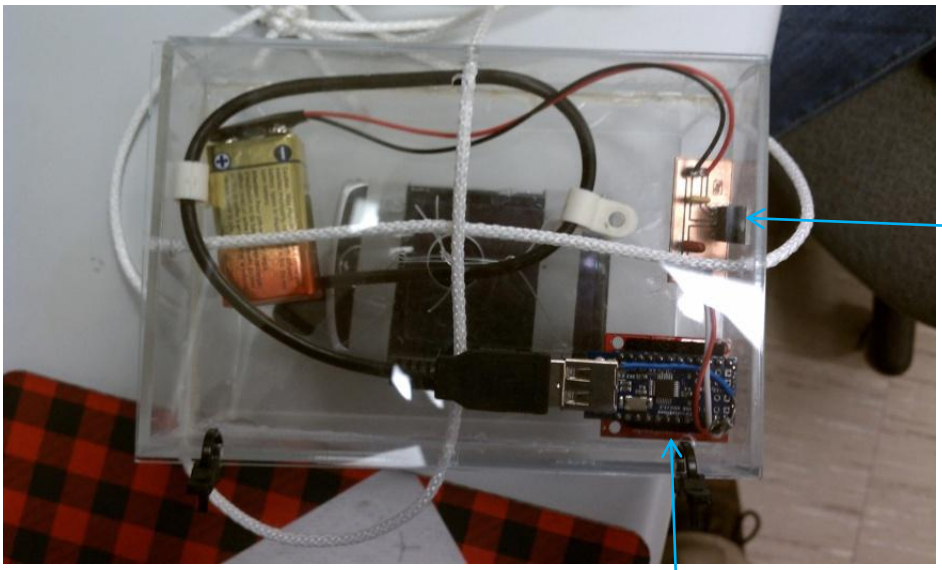


Figure 2: Enclosure - Top View

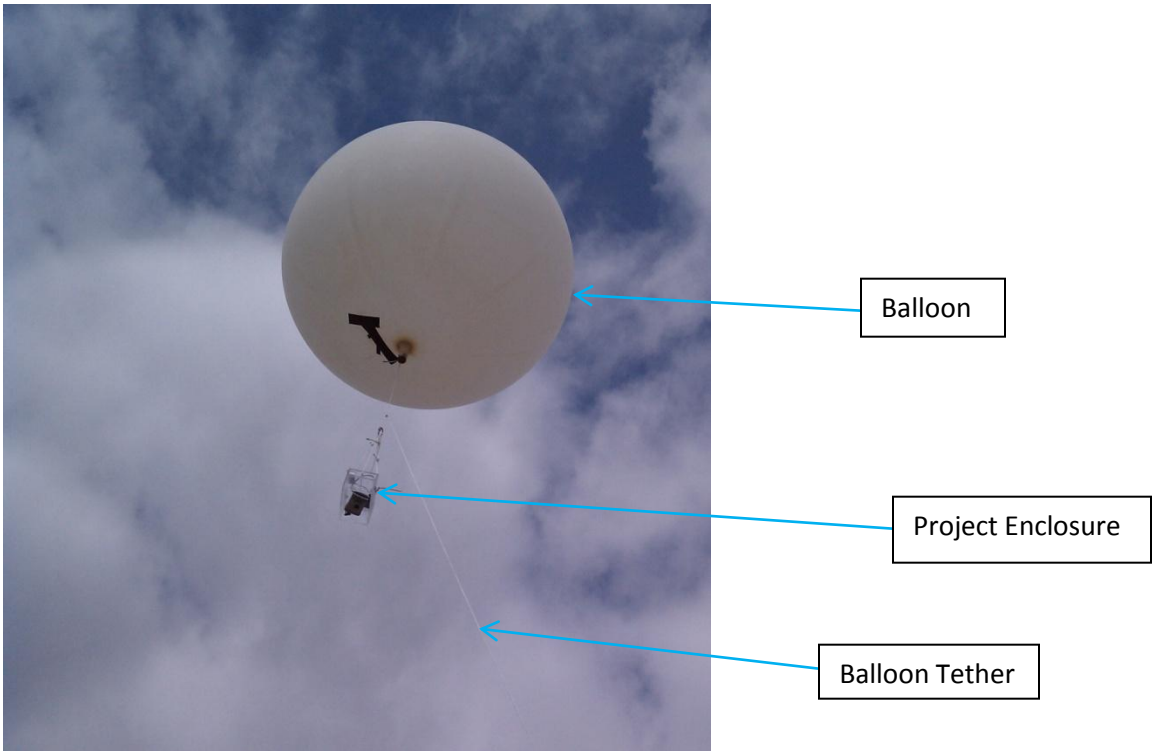


Figure 3: Project Airborne

Schematics

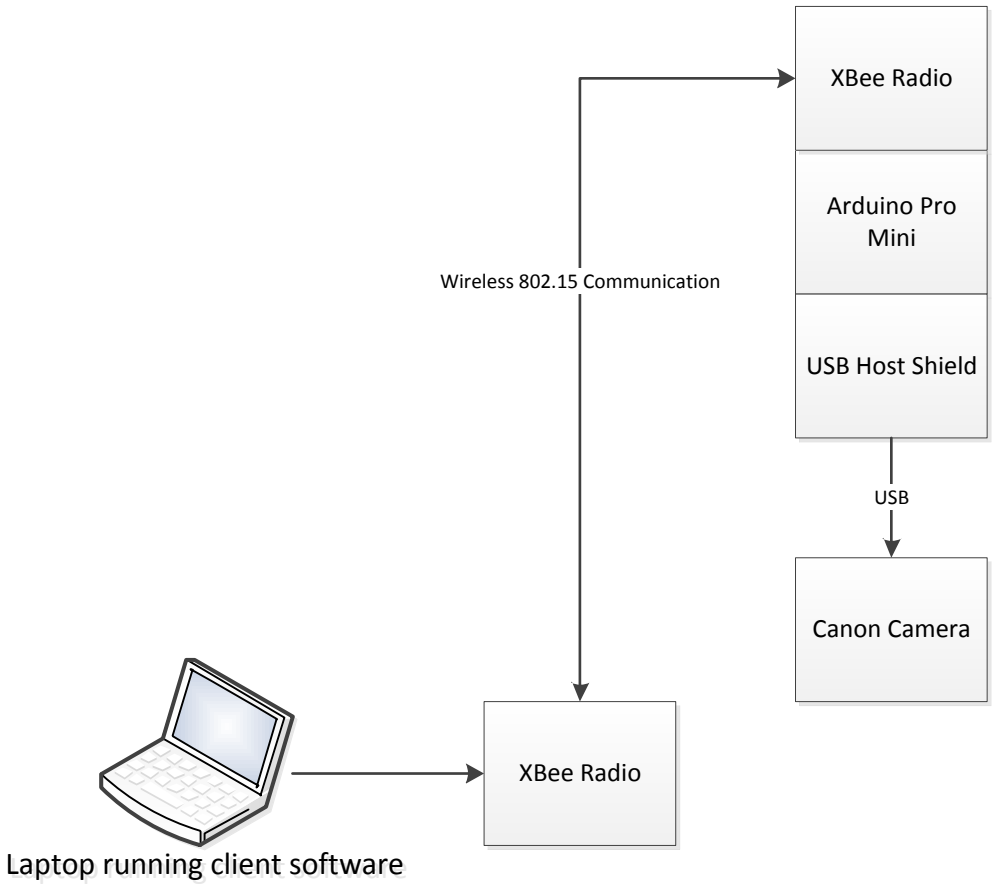


Figure 4: System Diagram

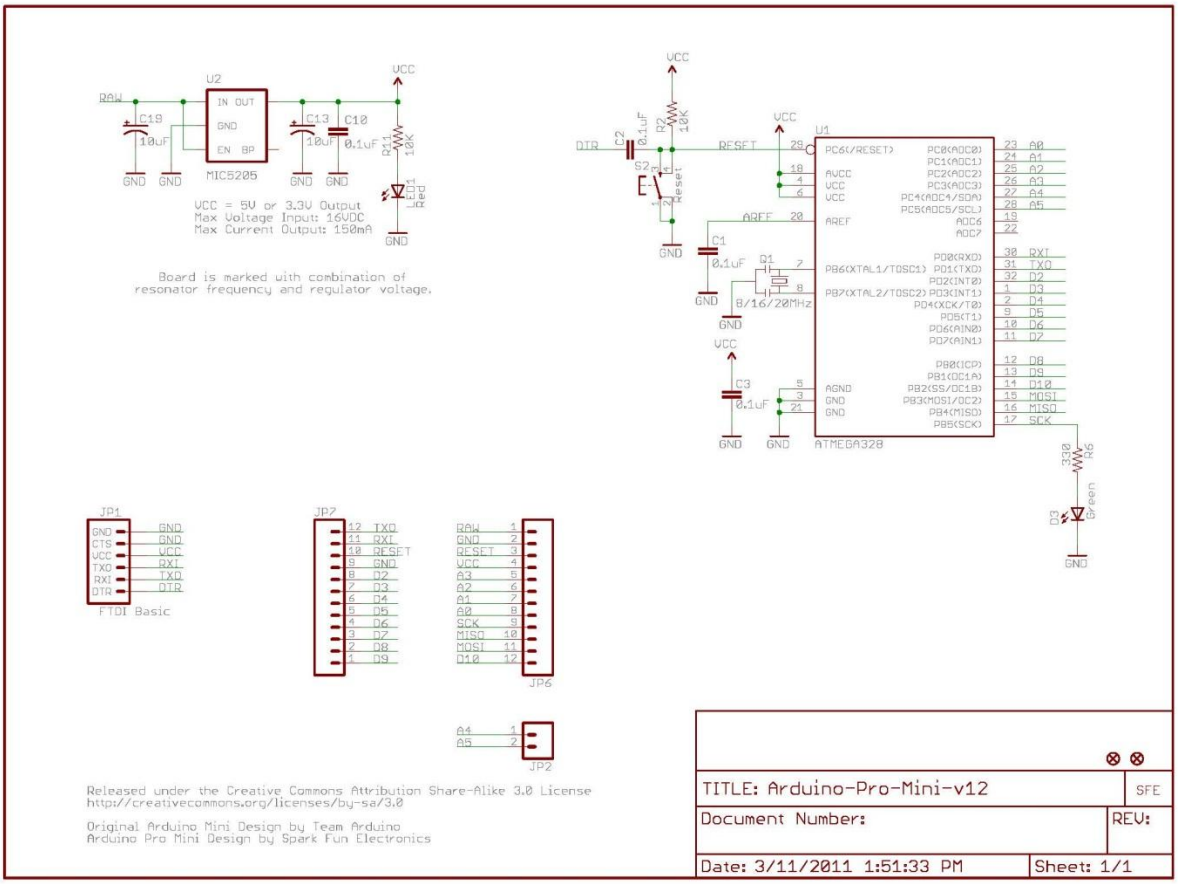


Figure 5: Arduino Pro Mini Schematic

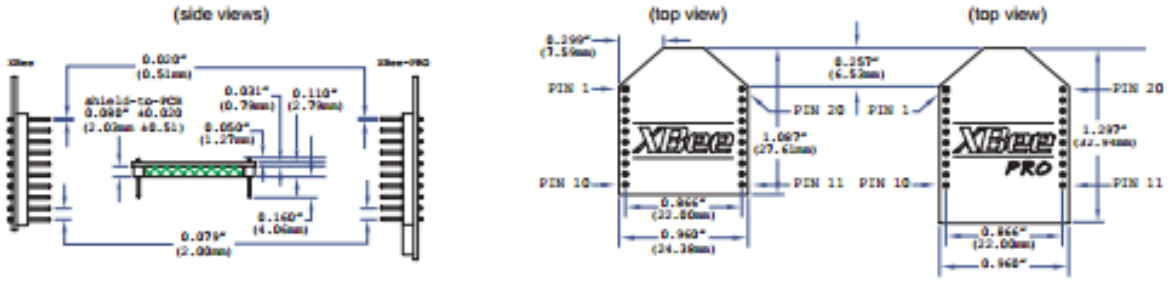


Figure 6: XBee Schematic

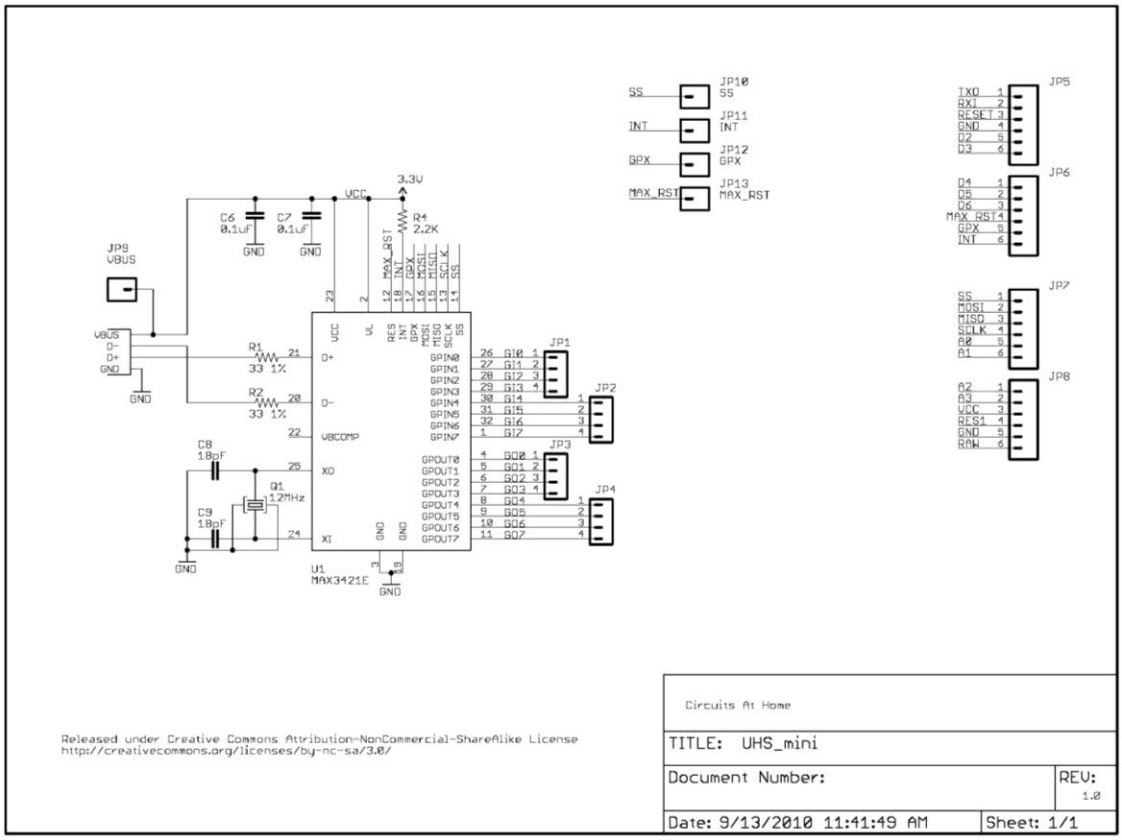


Figure 7: USB Shield for Arduino

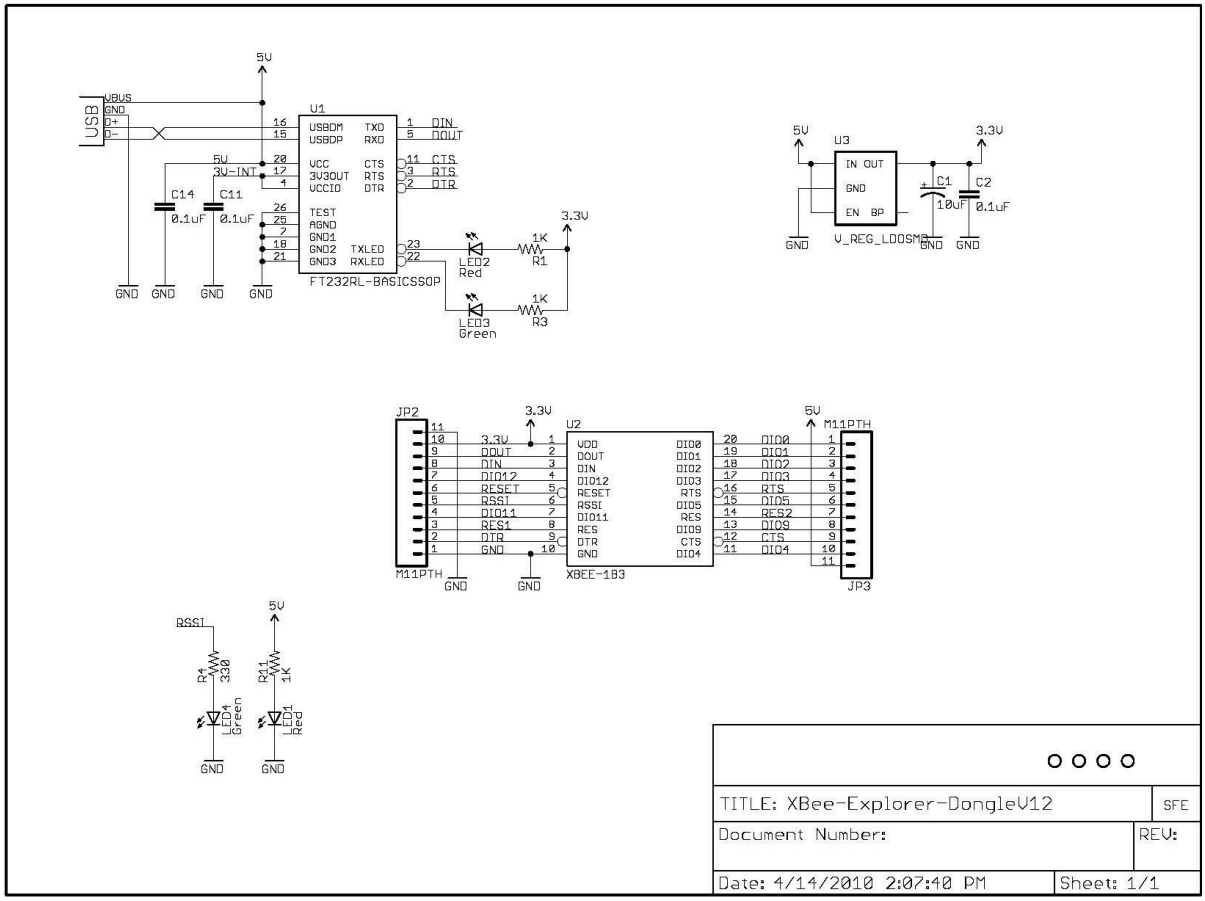


Figure 8: USB Shield for XBee

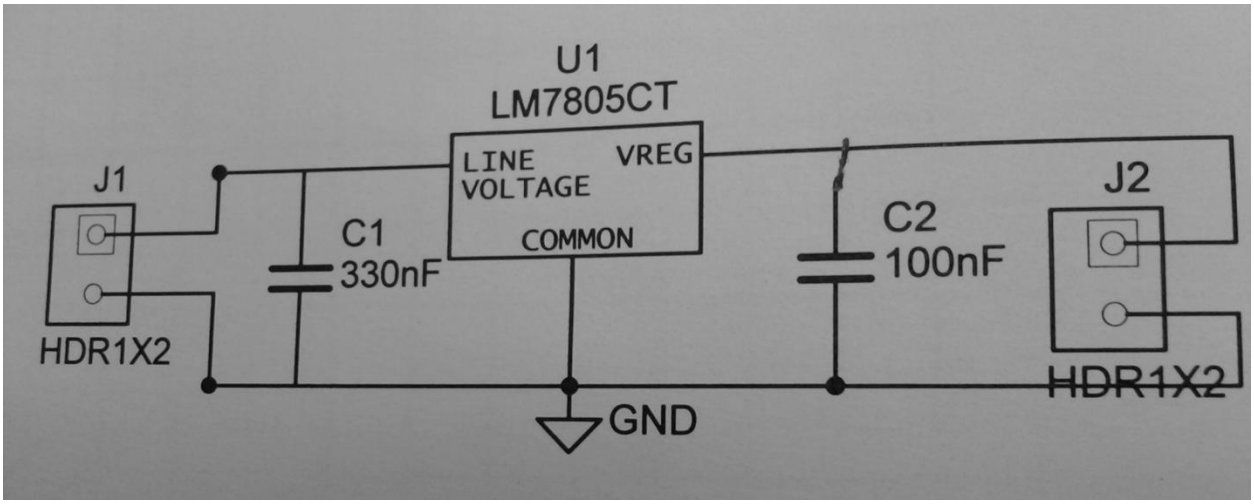


Figure 9: Voltage Regulator Schematic

Software

Flow Chart

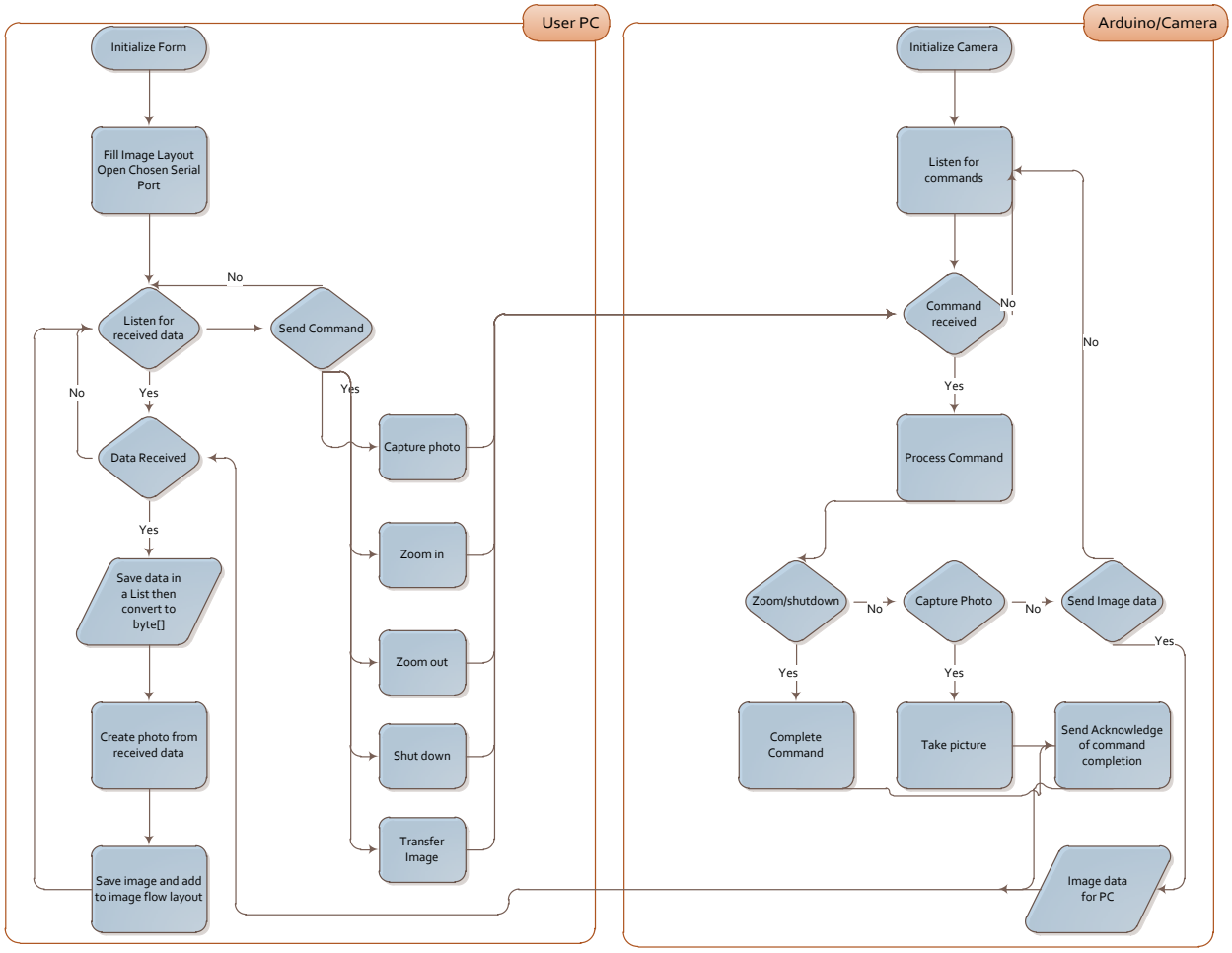


Figure 10: Process Flow Chart

Arduino Software

The Arduino microprocessor is used to enable communication with and control of the camera in the air. The Arduino's open-source compiler uses a language called Wiring, which is object-oriented and is very similar in syntax to C++. The compiler also includes C++ libraries, which allows for the inclusion of libraries written in C++. Because the platform is easy to use, open source, and available for low cost, Arduino-compatible software libraries are plentiful on the Internet.

Two libraries are used to facilitate communication between the Arduino and camera. The first is a library that abstracts the USB protocol and allows the Arduino to use the USB Host Shield. No modifications were made to code from this library and it will not be discussed here. The second library is an implementation of the Picture Transfer Protocol (PTP). PTP is a communications protocol meant to standardize the software interface presented by a digital imaging device to application and driver developers. Both of these software libraries are free and open source, licensed under the terms of the

GNU General Public License version 2 and were written and provided by the vendor of the USB Host Shield.

The Canon camera that we developed for does not include a full implementation of PTP. It is fully able to communicate using the PTP protocol, but it does not implement many of the operations specified by the PTP standard, including photo capture, zoom functions, and mode switching. In order to use these functions as specified in our requirements capture, we needed another software layer. This functionality was provided by the Canon Hack Development Kit (CHDK), an open-source third party firmware for Canon point-and-shoot cameras. Among other extensions of functionality, CHDK allows user-defined scripts (in the Lua programming language) to be run on the camera that allow full control of the camera, including simulated button presses and native access to various camera functions. CHDK runs concurrently with the camera's native firmware and allows normal camera operation while it is running. CHDK also includes a PTP interface so that software developers can communicate directly with a running CHDK instance on a camera remotely. This interface allows Lua scripts to be sent or triggered remotely, thereby granting access to the full functionality of the camera to any computer with a PTP implementation.

Implementation Details

The front-end of the Arduino's software is fairly simple. The Arduino maintains an event loop while waiting for a command to come from the serial port (The Xbee radios used by the project are seen by the Arduino as a normal serial port and require no software changes versus communicating with a computer). At the end of the event loop, the program maintains the USB connection and communicates with the serial port so that the client software knows it is still in communication. If no camera is connected, the event loop blocks and does not accept commands. A list of commands that the Arduino can interpret is given below:

Table 1: Camera Commands

Command	Action
C	Capture image
R	Switch to Record mode
P	Switch to Playback mode
Z	Zoom in
X	Zoom out
T	Transfer thumbnail of newest image
F	Transfer the newest image
S	Shutdown and power off the camera
D	Delete the newest image
B	Send the battery level of the camera

When a command is received from the client software, the Arduino passes that command or a series of commands to the camera. Most of the commands are straightforward and require only one camera operation. However, the thumbnail transfer and full picture transfer require multiple steps in order to execute correctly. In both cases, the Arduino must first determine which file is the newest so that users get the file they are expecting. First, a list of file handles is requested. A file handle is 32-bit unsigned

integer. Each file has a unique handle which is assigned when the file is created. The highest-numbered handle thus corresponds to the newest file. The Arduino then attempts to determine whether this file is an image or if it has a thumbnail. Before it can request file information for a specific file, the camera must be switched to Playback mode. If file information is requested while in Record mode, a picture thumbnail will not be generated and the thumbnail will not be available. Once the desired file is determined, the Arduino requests either the file or its thumbnail from the camera. The file is then dumped to the serial port for collection and reconstruction by the client.

Some additional backend software needed to be written in order to interface with CHDK. This is done with the CanonPSCHDK class. This class is an extension of the generic PTP class and includes replacement implementations of PTP operations not included with Canon cameras. All of the operations written make use of the PTP class's Transaction function. PTP operations have an opcode in addition to parameters and flags that specify the conditions with which the operation will run. These parameters are set and used to call Transaction, which then takes care of the protocol overhead required for communication. Optionally, Transaction allows the programmer to specify a parser to interpret data coming in from the camera. This parsing functionality is used to grab, store, and interpret information like file handles, file information, and the transfer of the file itself.

The CHDK PTP interface only specifies one opcode with the parameters specifying the actual operation. The only CHDK PTP operation implemented was the operation that allowed sending Lua scripts to be interpreted by the camera. Image capture, mode switching, zooming, and shutdown operations were each implemented by sending a line of Lua script to the camera. In addition to the additional operations provided, the CanonPSCHDK class implements a new initialization routine that uses the CHDK-enabled mode switching to put the camera into Record mode.

A list of files specific to the project is below. Note that not all library files are included in this list as it would make the list unreasonably long. The PTP and USB Host Shield libraries are required for the project to work. All files listed below except for cameraControlTest.pde have associated header files that contain function and constant definitions.

Table 2: CHDK Files

File Name	Description
cameraControlTest.pde	Main program, event loop and command interpreter
canonpschdk.cpp	Extension of the PTP class, includes CHDK-specific camera operations
chdkptp.h	Definition of the CHDK PTP interface including opcodes and parameter definitions
pseventparser.cpp	Handles camera states, such as connecting and disconnecting
ptpobjhandleparser.cpp	Parses object handles as they're delivered from the camera
ptpobjinfoparser.cpp	Parses object information as it's delivered from the camera
ptpthumbparser.cpp	Parser that takes the incoming file from the camera and outputs it to the serial port

Maintenance

The Xbee radio used to communicate wirelessly is soldered onto the terminals used by the Arduino to communicate with a PC over serial. The Xbee cannot be used to transfer code to the Arduino. Thus, no software can be uploaded to the Arduino. The current software load should be considered the final version.

PC User Software

The PC software is the interface that the customer will use to control the camera remotely. The PC software was built on the Microsoft WinForms, using C#. This was chosen due to the easy implementation of creating a user interface and overlaying that with built-in serial communications classes that were used to communicate with the camera. There are multiple windows implemented in the program. Two are used for the purpose of communicating with the camera and to examine photos, while two others are for informational purposes.

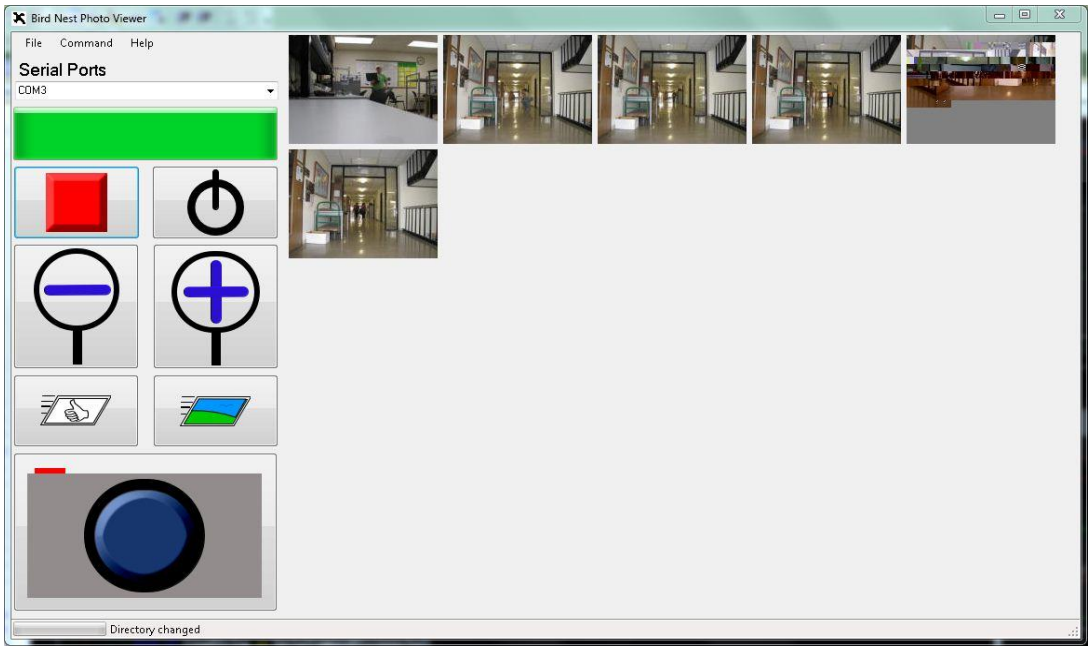


Figure 11: User Interface Main Window

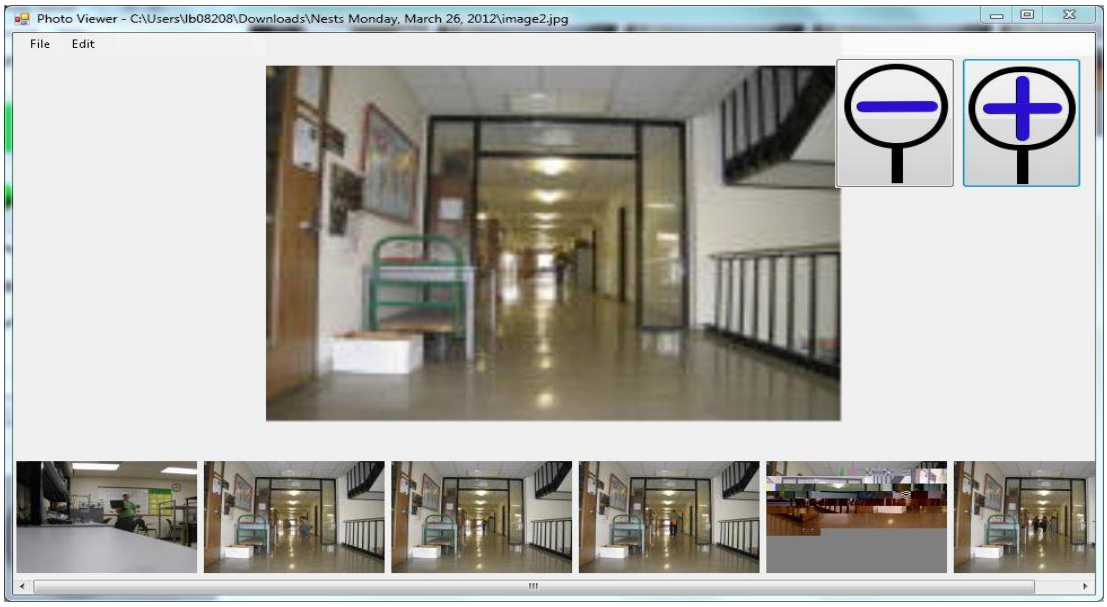


Figure 12: User Interface Photo Viewer

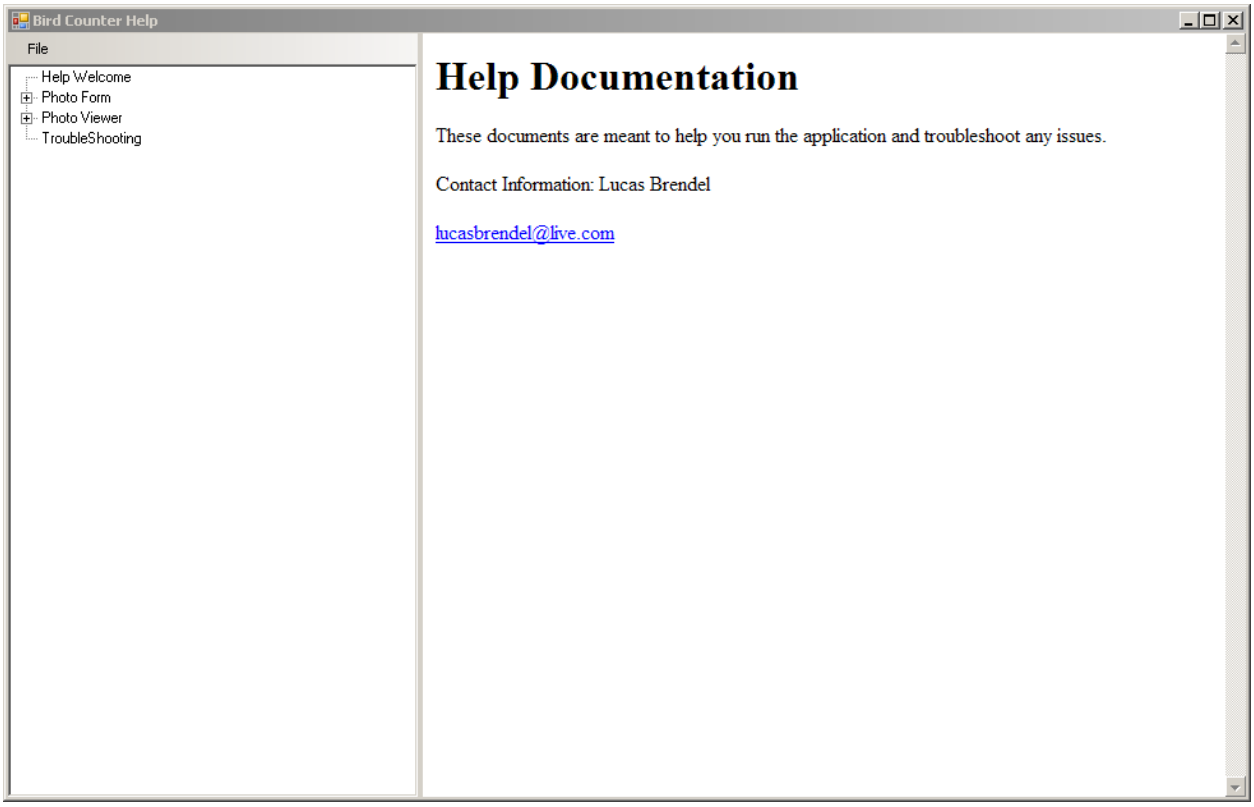


Figure 13: Help Documentation Window

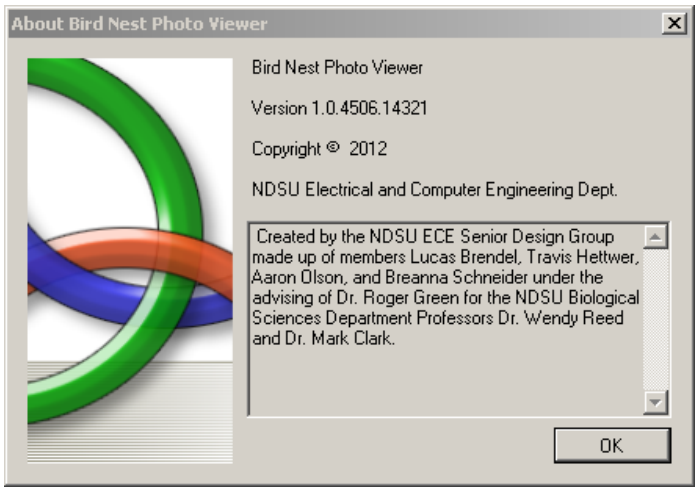


Figure 14: About Box

The main window in FIGURE 11 contains buttons for all the commands the user can use to control the camera. On the right it contains a photo viewing pane that displays all the photos the user has transferred from the camera. The photo viewer in FIGURE 12 displays all folders that the user has transferred to the camera. Photos can be increased or decreased in size for easier viewing.

To initiate communication with the camera the user has to open the correct serial port. A drop down shows all available serial ports on the machine. The commands that the PC interface sends to the camera are the same as explained in TABLE 1. Each command is sent when the respective button is clicked.

Once the serial port is opened and communication with the camera is initiated commands can be sent. The serial port has an event listener for anything sent to the XBee. When a message is received a "Data Received" event is triggered and the serial port code processes the messages to determine the code. If what was received matches a message code then the interface processes the bytes following the message accordingly. The flowchart below shows the operations the serial port can go through.

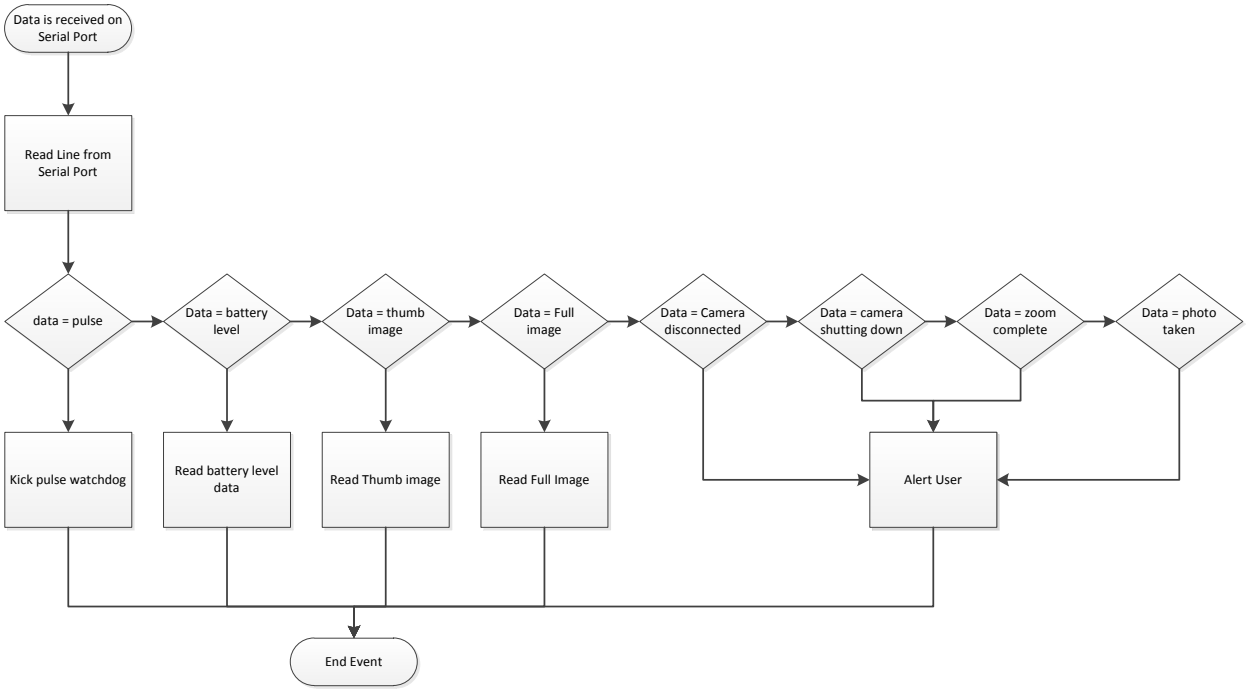


Figure 15: Serial Data Received Flow Chart

Most messages received have a display message for the user that they can see what the serial port is processing. So when a thumbnail is being transferred the status message will display a message explaining that it is processing a photo. Due to the rate of processing commands, some messages are not always visible when being processed. When the serial port is not processing specific commands it is listening for the Arduino's pulse that is sent. This pulse ensures that connections are still made with the camera. TABLE 3 shows all command messages the user could see.

Table 3: Serial Processing Messages

Command Message	Occurrence
Camera Connected	When there exists communication with the camera.
Picture Transfer Start	When the user requests a full photo and the camera acknowledges.
Thumbnail Transfer Start	When the user requests a thumbnail photo and the camera acknowledges
Command Received. Triggering capture!	The camera acknowledges the command and is capturing a photo
Command received. Zooming out	The camera received command and is zooming out
Command received. Zooming in	The camera received command and is zooming in
Command received. Shutting down!	The camera is shutting down
Camera disconnected	The pulse that the camera sends has not been received for an extended period of time and more than likely has lost connection with the PC.
Command received. Sending battery level	The camera is sending the battery level.

The most involved commands are the photo and thumbnail transfer commands. These require careful processing to ensure that all the bytes have been received and processed. When the serial port receives a thumbnail/photo transfer command, the next command it receives is the size of the photo. After that the serial port reads for all bytes for the image. Each byte it receives it places in to a List. The list is then converted to a byte array and then placed in to a memory stream which creates an image out of the bytes. The memory stream is very picky and needs to have all correct byte flags to successfully create an image. JPG images have certain strings of bytes that signal a specific portion of the message. Since the main concern is just to transfer the image and not to compress it or alter it, all the serial port has to look for is the start and end key for the image. The start of an image is 0xFF 0xD8. Once that has been read the serial port will start saving the bytes following it. The end key is 0xFF 0xD9. After this sequence has been received the list will then be converted to an image. The flow chart in FIGURE 16 shows this process.

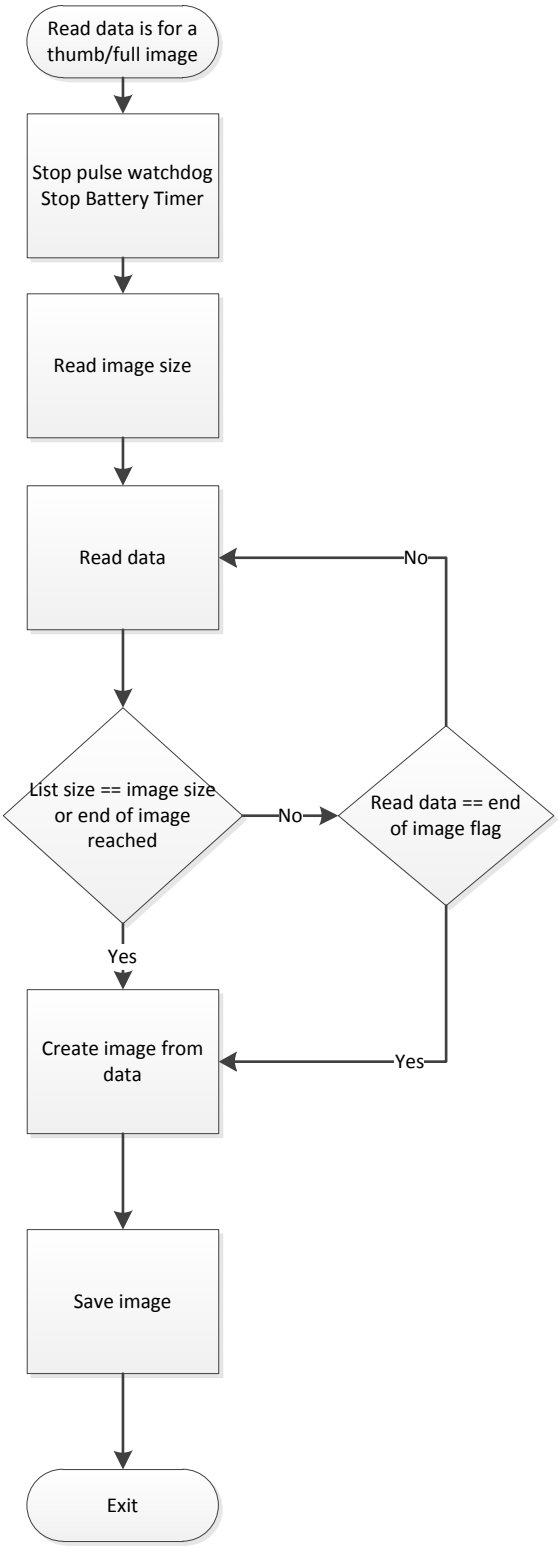


Figure 16: Image Processing Flow Chart

The software is broken in to four sets of files. Each set corresponds to window of the interface. TABLE 4 gives a breakdown of the files. Each file has 3 3 other files attached to it. *filename.Designer.cs, filename.resx,* and *filename.cs [Design].* These contain build information about the controls used and other resources used.

Table 4: User Interface code

Files	Purpose
PhotoForm.cs	Contains interface code for the main window.
PhotoViewer.cs	Contains interface code for viewing photos.
HelpForm.cs	Contains interface code for displaying help documentation
AboutBox.cs	Contains assembly information about the project

Technician's Troubleshooting Section

Problem: Balloon not lifting payload or not lifting higher than a certain height

The amount of weight the balloon can lift is directly related to how much helium is in the balloon. If the balloon is not lifting the payload, add more helium. Keep in mind that as the balloon rises, it has to lift more and more weight as the tether extends.

Problem: Camera not connected to client software

First, ensure that the camera is on. Next, make sure that the Arduino and radio are getting power. Both of these components should have power LEDs to indicate that they are powered on. If the LEDs are not on, make sure that both the battery and the Arduino are connected to the voltage regulator board. These connections coming loose is the most common cause of the Arduino losing power. If all of the connections have been made, it is possible that the battery is dead. In that case, replace the battery and try again.

If the power LEDs are on and the camera still does not connect, first make sure that the XBee chip on the Arduino is pointing in the general direction of the XBee chip attached to the computer. The antennas are somewhat directional and the signal may not be received if they are pointed away from each other. If repositioning the antennas does not solve the problem, try turning the camera off and then on again. Additionally, if the camera does not connect, this may be an indication that the battery is dying. A dying battery may be able to power up the Arduino and XBee, but not provide enough voltage to activate the camera.

Problem: Camera doesn't respond and appears frozen

Occasionally, the camera may lock up. While this should not happen in normal use, it is possible. The camera may be locked up to the point where it does not respond to button presses. In this case, remove the camera's battery, reinsert it, and try again.

Problem: Corrupted or incomplete image received

At long distances (greater than 100 feet), occasionally thumbnails or images will not be received correctly, resulting in image corruption. To resolve the problem, try sending the picture again. Make sure the laptop's XBee module is pointed at the camera enclosure.

Project Comments

Project Issues

Our project went fairly smoothly as far as technical issues were concerned. The only issues we had were fairly minor. We had some problem with image corruption during transfer and with communication range with the Arduino. The image corruption issue appears to be due to random noise or other interference from external sources and only occurs at distances greater than 100 feet. The communications issues are also range related.

Some minor issues were encountered with the thumbnail transfer functionality. A specific sequence of commands had to be followed in order to successfully request the thumbnail from the camera. Incorrect command sequences either resulted in the camera locking up, which required a hard reset, or the camera not generating the thumbnail, thus making thumbnail transfer impossible. By requesting a list of files, switching to playback mode, then requesting the thumbnail, we were able to successfully request and transfer thumbnails.

We also had issues with trying to test the project in the air. The first balloon we purchased was found to have holes in it after we filled it with helium. This balloon later popped when we tried patching the holes. The second balloon we tested was from the department's stock. We found that this balloon also had holes, but we were able to successfully patch them to conduct a test. We found that local wind conditions made it difficult to test the balloon. Buildings and other ground structures cause turbulent wind conditions near the ground, which made the first 30 feet or so of airspace challenging for the balloon.

Lessons Learned

We learned many things this semester, including:

- Time management is tremendously important and can really relieve stress when due dates approach.
- A divide-and-conquer strategy is the most effective for completing a large project like this.
- Additional prototype tests would have been very useful to conduct. Because we are limited by weather conditions, it can be very difficult to find good testing times. Nonetheless, we should have tested earlier and more often.
- Budgets are important and it is paramount to find solutions that fit into your budget.
- We learned how to evaluate a set of requirements and come up with a solution that completely meets them.
- Bre makes really good cupcakes. Also, she tends to break things.

Future Work

This project has room for future additions. We discussed performing image processing on the images taken by the airborne camera. This image processing would ideally be able to identify nests from the images, count them, and perform statistical areal analysis, thereby greatly aiding the data collection efforts. Additionally, we discussed adding a GPS device to the project so that the clients would be able to tell exactly where the images were taken.

Further revision could be done on the mounting system for the camera. Our current mounting system uses gravity to keep the camera pointed straight down. However, this setup is susceptible to high winds and requires careful balancing during setup. A revised camera mount could use a gimbal system to keep the camera pointed down. This would be much more robust and likely easier to setup. We researched commercial systems to do this, but they tended to be expensive and heavy. A purpose-designed gimbal could have the potential to be lightweight and cheap enough to be effective in this situation.

Future projects could make use of different aerial vehicles. We chose a balloon due to low cost and ease of use, but other, more expensive alternatives exist. For example, quadrocopters have been used to take aerial images and have even seen use in the movie industry. A quadrocopter would be more expensive than a balloon, but it would be much easier to pilot than an airplane or helicopter.

Appendix A: PCB Layouts

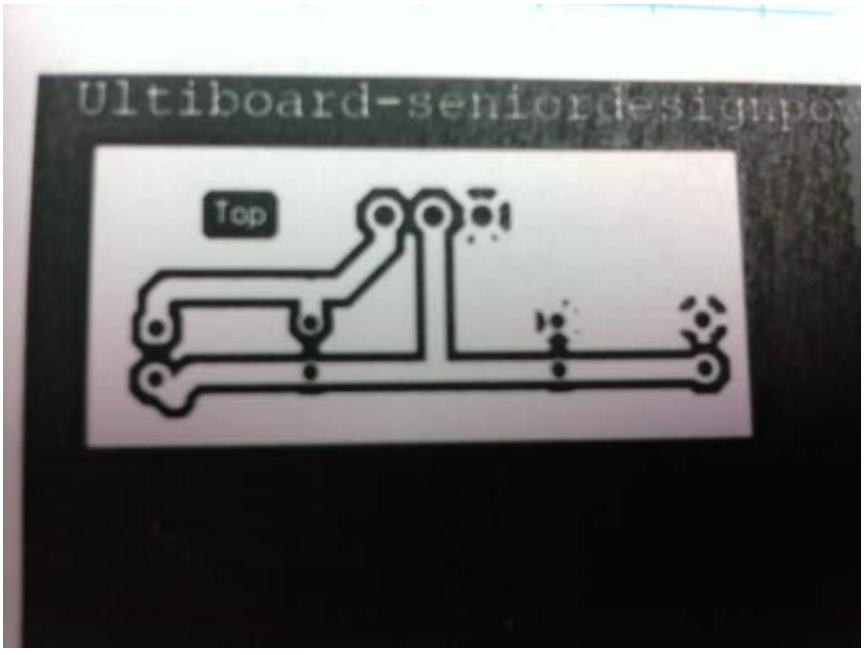


Figure 17: Voltage Regulator Circuit Layout (Top)

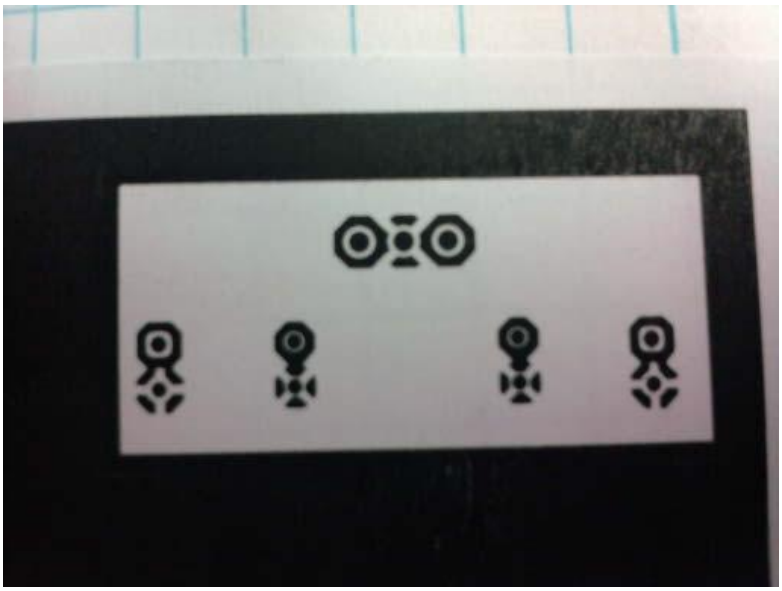


Figure 18: Voltage Regulator Circuit Layout (Bottom)

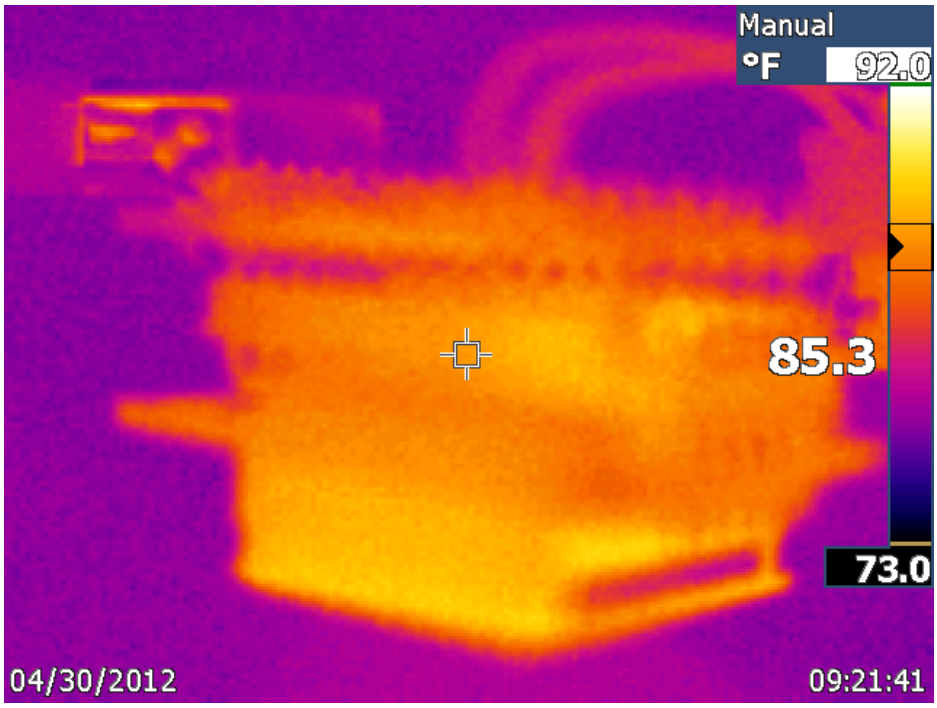


Figure 19: Arduino/XBee/USB Shield Thermal Image

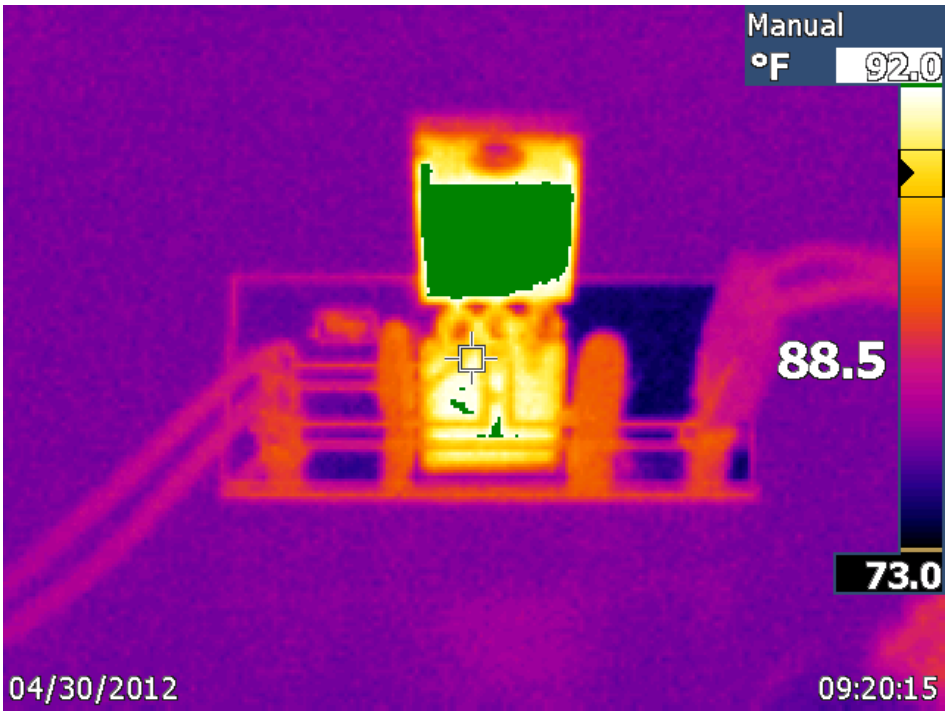


Figure 20: Voltage Regulator Thermal Image

Appendix B: Parts List and Budget

Table 5: Budget

Item	Unit Cost ($)	Quantity	Paid($)	Total ($)
Weather Balloon	34	2	68	68
XBee Wireless Antenna	22.95	2	45.9	113.9
Break-out board for XBee	9	1	9	122.9
Ardunio Pro Mini 3.3V	18.95	1	18.95	141.85
USB host shield for Ardunio	25	1	25	166.85
XBee Explorer Dongle	25	1	25	191.85
Balloon Tether	0.22	200 ft	44	235.85
Helium	125	300 cu ft	125	360.85
Helium Tank Rental	7	2 months	14	374.85
Epoxy	6.5	1	6.5	381.35
Winch	35	1	35	416.35
PCB	33	1	0	416.35
9V Batteries	2	2	0	416.35
Plexiglass	8.59	5 sq ft	0	416.35
Fishing Line	5.5	15 ft	0	416.35
USB Mini Cable	5	1	0	416.35
LM7805	0.69	1	0	416.35
Laser Cutting Services	10	2 hrs	0	416.35
Acessories for Enclosure	3	6	0	416.35

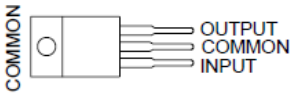
Appendix C: Data Sheets

μA7800 SERIES
POSITIVE-VOLTAGE REGULATORS

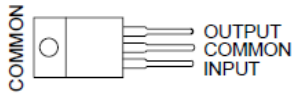
SLVS056J – MAY 1976 – REVISED MAY 2003

- 3-Terminal Regulators
- Output Current up to 1.5 A
- Internal Thermal-Overload Protection
- High Power-Dissipation Capability
- Internal Short-Circuit Current Limiting
- Output Transistor Safe-Area Compensation

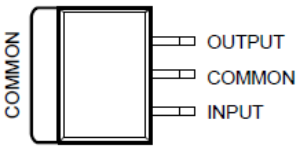
KC (TO-220) PACKAGE
(TOP VIEW)



KCS (TO-220) PACKAGE
(TOP VIEW)



KTE PACKAGE
(TOP VIEW)



description/ordering information

This series of fixed-voltage integrated-circuit voltage regulators is designed for a wide range of applications. These applications include on-card regulation for elimination of noise and distribution problems associated with single-point regulation. Each of these regulators can deliver up to 1.5 A of output current. The internal current-limiting and thermal-shutdown features of these regulators essentially make them immune to overload. In addition to use as fixed-voltage regulators, these devices can be used with external components to obtain adjustable output voltages and currents, and also can be used as the power-pass element in precision regulators.

ORDERING INFORMATION

T_J	$V_{O(NOM)}$ (V)	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 125°C	5	POWER-FLEX (KTE)	Reel of 2000	μA7805CKTER	μA7805C
		TO-220 (KC)	Tube of 50	μA7805CKC	μA7805C
		TO-220, short shoulder (KCS)	Tube of 20	μA7805CKCS	
	8	POWER-FLEX (KTE)	Reel of 2000	μA7808CKTER	μA7808C
		TO-220 (KC)	Tube of 50	μA7808CKC	μA7808C
		TO-220, short shoulder (KCS)	Tube of 20	μA7808CKCS	
	10	POWER-FLEX (KTE)	Reel of 2000	μA7810CKTER	μA7810C
		TO-220 (KC)	Tube of 50	μA7810CKC	μA7810C
	12	POWER-FLEX (KTE)	Reel of 2000	μA7812CKTER	μA7812C
		TO-220 (KC)	Tube of 50	μA7812CKC	μA7812C
		TO-220, short shoulder (KCS)	Tube of 20	μA7812CKCS	
	15	POWER-FLEX (KTE)	Reel of 2000	μA7815CKTER	μA7815C
		TO-220 (KC)	Tube of 50	μA7815CKC	μA7815C
		TO-220, short shoulder (KCS)	Tube of 20	μA7815CKCS	
	24	POWER-FLEX (KTE)	Reel of 2000	μA7824CKTER	μA7824C
		TO-220 (KC)	Tube of 50	μA7824CKC	μA7824C

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Figure 21: LM8705 Voltage Regulator Data Sheet

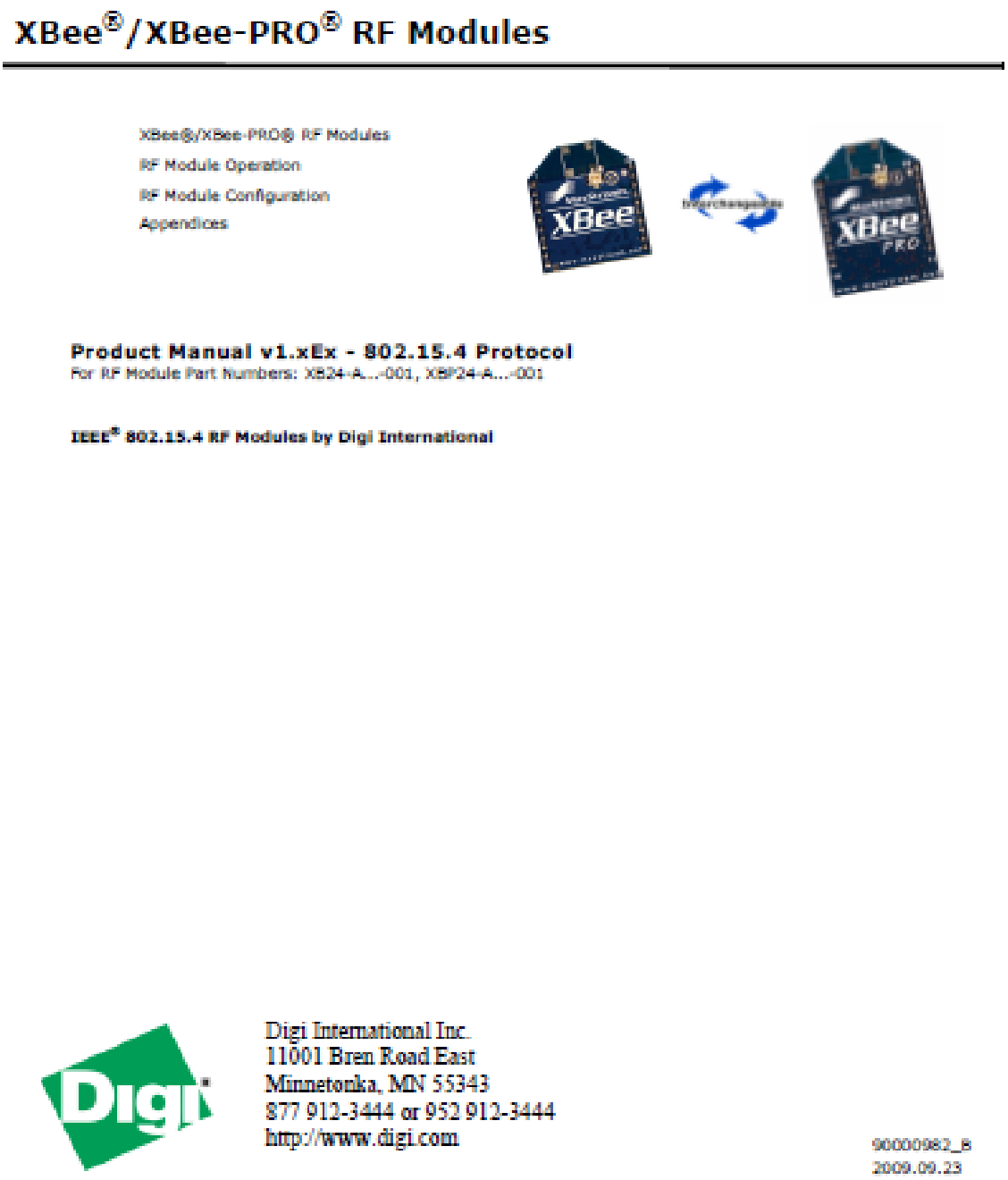


Figure 22: XBee Datasheet

Arduino Pro Mini 328 - 5V/16MHz

sku: DEV-09218 RoHS Compliant

Replacement: DEV-11113. The new version of the Pro Mini breaks out the ADC 6 and 7 pins! This page is for reference only.

Description: It's blue! It's thin (0.8mm)! It's the Arduino Pro Mini! SparkFun's minimal design approach to Arduino. This is a 5V Arduino running the 16MHz bootloader (select 'Arduino Duemilanove w/ 328' within the Arduino software). Arduino Pro Mini does not come with connectors populated so that you can solder in any connector or wire with any orientation you need. We recommend first time Arduino users start with the Uno R3. It's a great board that will get you up and running quickly. The Arduino Pro series is meant for users that understand the limitations of system voltage (5V), lack of connectors, and USB off board.

We really wanted to minimize the cost of an Arduino. The Arduino Pro Mini is like the Arduino Mini (same pin out) but to keep the cost low, we used all SMD components, made it two layer, etc. This board connects directly to the FTDI Basic Breakout board and supports auto-reset. The Arduino Pro Mini also works with the FTDI cable but the FTDI cable does not bring out the DTR pin so the auto-reset feature will not work.

Note: A portion of this sale is given back to Arduino LLC to help fund continued development of new tools and new IDE features.

Features:

- ATmega328 running at 16MHz with external resonator (0.5% tolerance)
- USB connection off board
- Supports auto-reset
- 5V regulator
- Max 150mA output
- Over current protected
- Reverse polarity protected
- DC input 5V up to 12V
- On board Power and Status LEDs

Dimensions:

- 0.7x1.3" (18x33mm)
- Less than 2 grams

USB Host Shield (no data sheet, beginning of hardware manual is provided here)

1. Introduction

USB Host Shield is an inexpensive ($25 for the full-sized board and $20 for the Mini variant) add-on board for Arduino development platform. The shield provides USB Host interface, allowing full and low-speed communication with USB devices – keyboards, mice, joysticks, MIDI, digital cameras, Bluetooth, and many others. In USB Shield section of this site you can find many articles describing projects and code examples written for this shield. On this page, I'm giving detailed description of board's hardware. I start with explaining board's connectors, pads and jumpers, as well as differences between shield variants. Finally, I demonstrate ways to adapt USB Host Shield to non-typical Arduino boards and less-common power configurations.

This document covers both full-size and Mini shield variants. At the time of writing, current revision of full size USB Host Shield is 2.0 and current Mini shield is 1.1. Older revisions of the shield will be described later.

Appendix D: Software Code